
Armors Labs
BBC EXCHANGE TRX
Smart Contract Audit

BBC EXCHANGE Audit Summary

Project name : BBC EXCHANGE for Trx.

Project address: None

Code URL : None

Projct target : Exchange Contract Audit

Test result : The project is a decentralized cross chain exchange

The contract basically meets the specifications, the code quality is relatively high.

Audit Info

Audit NO : 0X202010190001

Audit Team : Armors Labs

BBC EXCHANGE Audit

The BBC Group team asked us to review and audit their EXCHANGE contract. We looked at the code and now publish our results.

Here is our assessment and recommendations, in order of importance.

Document information

Name	Auditor	Version	Date
BBC EXCHANGE Audit	Rock ,Hosea, Rushairer	1.0.0	2020-10-19

Review

Audit results

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the BBC EXCHANGE contract. The above should not be construed as investment advice.

Based on the widely recognized security status of the current underlying blockchain and smart contract, this audit report is valid for 18 months from the date of output.

(Statement: Armors Labs reports only on facts that have occurred or existed before this report is issued and assumes corresponding responsibilities. Armors Labs is not able to determine the security of its smart contracts and is not responsible for any subsequent or existing facts after this report is issued. The security audit analysis and other content of this report are only based on the documents and information provided by the information provider to Armors Labs at the time of issuance of this report (" information provided " for short). Armors Labs postulates that the information provided is not missing, tampered, deleted or hidden. If the information provided is missing, tampered,

deleted, hidden or reflected in a way that is not consistent with the actual situation, Armors Labs shall not be responsible for the losses and adverse effects caused.)

Audited target file

file	md5
BBC_TRX.sol	94a609467f8d5d9e63c1946a116d0f52

Vulnerability analysis

Vulnerability distribution

vulnerability level	number
Critical severity	0
High severity	0
Medium severity	0
Low severity	0

Summary of audit results

Vulnerability	status
Re-Entrancy	safe
Arithmetic Over/Under Flows	safe
Unexpected Ether	safe
Delegatecall	safe
Default Visibilities	safe
Entropy Illusion	safe
External Contract Referencing	safe
Short Address/Parameter Attack	safe
Unchecked CALL Return Values	safe
Race Conditions / Front Running	safe
Denial Of Service (DOS)	safe
Block Timestamp Manipulation	safe
Constructors with Care	safe
Unintialised Storage Pointers	safe
Floating Points and Numerical Precision	safe
tx.origin Authentication	safe

Contract code

```
pragma solidity >= 0.5.10;

interface IERC20 {
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint256);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

library SafeMath {

    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }


    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }


    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
       if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }


    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }


    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }
```

```
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }


    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != accountHash && codehash != 0x0);
    }

    /**
     * @dev Converts an `address` into `address payable`. Note that this is
     * simply a type cast: the actual underlying value is not changed.
     *
     * _Available since v2.4.0._
     */
    function toPayable(address account) internal pure returns (address payable) {
        return address(uint160(account));
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects
     *
```

```
     * _Available since v2.4.0._
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-call-value
        (bool success, ) = recipient.call.value(amount)("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }
}

library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value
    }

    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowanc
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreas
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowanc
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxin
     * on the return value: the return value is optional (but if data is returned, it must not be fal
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking m
        // we're implementing it ourselves.

        // A Solidity high level call has three parts:
        //  1. The target address is checked to verify it contains contract code
        //  2. The call itself is made, and success asserted
        //  3. The return value is decoded, which in turn checks the size of the returned data.
        // solhint-disable-next-line max-line-length
        // require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
```

```
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}

contract Context {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.co
        return msg.data;
    }
}

contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view returns (address) {
        return _owner;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev Returns true if the caller is the current owner.
     */
    function isOwner() public view returns (bool) {
        return _msgSender() == _owner;
    }

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public onlyOwner {
```

```
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

contract BBCExchange is Ownable {

    using SafeMath for uint256;
    using SafeERC20 for IERC20;
    // event
    event createOrderEvent(address sender, address sellToken, uint256 sellAmount,
            string buyToken, uint256 buyAmount, uint256 fee, address goBetween,
            string carryOut, uint256 blockHeight);
    event retraceEvent(address indexed _to, uint256 _amount);
    event withdrawEvent(address indexed _to, uint256 _amount);
    event betweenEvent(address orderAddress, address sellToken,string buyToken, uint256 amount, addre
            address carryOutAddress, bytes32 randIHash, bytes32 randJHash, string randKey);
    event debug(uint256 msg);

    //struct
    struct TradePair {
        address sellToken;
        uint256 sellAmount;
        string buyToken;
        uint256 buyAmount;
    }

    struct Operate{
        address goBetween;//撮合地址
        string carryOut;//执行地址
    }

    struct GoBetween {
        uint256 sellAmount;//撮合数量
        uint256 buyAmount;//兑换数量
        address payable buyAddress;//买入者钱包地址
        address carryOut;//乙方执行者地址
        bytes32 randIHash;//随机数I的Hash
        bytes32 randJHash;//随机数J的Hash
        string randKey;//随机数秘值
        bytes32 primaryKey;//唯一主键
        uint256 blockNo;//撮合时块高度
    }

    struct Order {
        mapping(string => TradePair) tradePair;
        mapping(string => Operate) operate;
        mapping(bytes32 => GoBetween) betweens;
        mapping(bytes32 => bytes32[]) betweensKeys;
```

```
        address owner;//所有者
        string ownerOtherWalletAddress;//挂单者在另外链上的钱包地址
        uint256 fee;//手续费
        uint256 blockHeight;//交易限制块高度
        bytes32 primaryKey;//唯一主键
    }



    mapping (address => mapping(bytes32 => Order)) public orderList;


   function getOrderTradePair(address _user, bytes32 _primaryKey) public view returns(
        address sellToken,
        uint256 sellAmount,
        string memory buyToken,
        uint256 buyAmount){
            TradePair memory _tradePair = orderList[_user][_primaryKey].tradePair["trade"];
            return(_tradePair.sellToken, _tradePair.sellAmount, _tradePair.buyToken, _tradePair.buyAm
    }

    function getOrderOperate(address _user, bytes32 _primaryKey) public view returns(
        address goBetween,
        string memory carryOut){
            Operate memory _operate = orderList[_user][_primaryKey].operate["operate"];
            return(_operate.goBetween, _operate.carryOut);
    }

    function getBetween(address _user, bytes32 _orderPrimaryKey, bytes32 _betweenPrimaryKey) public v
            bytes32 randIHash,bytes32 randJHash,string memory randKey, uint256 buyAmount){
        GoBetween memory between = orderList[_user][_orderPrimaryKey].betweens[_betweenPrimaryKey];
        return (between.sellAmount, between.buyAddress, between.carryOut,
                between.randIHash,between.randJHash,between.randKey,between.buyAmount);
    }

    /**
     * 创建挂单
     * 参数参考struct Order
     * 需要预授权
     */
    function createOrder(address sellToken, uint256 sellAmount,
            string memory buyToken, uint256 buyAmount, uint256 fee, address goBetween,
            string memory carryOut, uint256 blockHeight, string memory ownerOtherWalletAddress,
            bytes32 primaryKey) public payable {
        require( sellAmount > 0,"createOrder:sellAmount invalid");
        require( buyAmount > 0,"createOrder:buyAmount invalid");
        require( goBetween != address(0),"createOrder:goBetween invalid");
        require(orderList[msg.sender][primaryKey].blockHeight == 0,"createOrder: Order primaryKey is
        require(fee <= 10000 && fee > 0,"createOrder: fee less than 10000 and more than zero");

        if ( msg.value > 0 ) {
            sellToken = address(0);
            sellAmount = msg.value;
        }else{
            IERC20(sellToken).safeTransferFrom(msg.sender, address(this), sellAmount);
        }

        orderList[msg.sender][primaryKey] = Order({owner:msg.sender, fee:fee, blockHeight:blockHeight
                ownerOtherWalletAddress:ownerOtherWalletAddress, primaryKey:primaryKey});
        orderList[msg.sender][primaryKey].tradePair["trade"] = TradePair(sellToken, sellAmount, buyTo
        orderList[msg.sender][primaryKey].operate["operate"] = Operate(goBetween, carryOut);
        emit createOrderEvent(msg.sender, sellToken,sellAmount,buyToken,buyAmount,fee,goBetween,carry
    }


    //  /**
```

```
    //  * 撮合挂单
    //  * orderAddress     挂单钱包地址
    //  * sellToken        卖出合约地址
    //  * buyToken         卖入合约地址
    //  * 参数参考struct GoBetween
    //  */
     function goBetween(address orderAddress, address sellToken, uint256 amount, string memory buyTok
            address carryOutAddress, bytes32 randIHash, bytes32 randJHash, string memory randKey,byte
            //有可撮合的额度 && 当前块高度小于指定指定高度 && 当前用户是指定的撮合者
            require(orderList[orderAddress][ordrePrimaryKey].tradePair['trade'].sellAmount >= amount,
            require(orderList[orderAddress][ordrePrimaryKey].operate["operate"].goBetween == msg.send
            require(orderList[orderAddress][ordrePrimaryKey].betweens[betweenPrimaryKey].carryOut ==
             //验证交易对
            require(orderList[orderAddress][ordrePrimaryKey].tradePair['trade'].sellToken == sellToke
            require(keccak256(abi.encodePacked(orderList[orderAddress][ordrePrimaryKey].tradePair['tr

             //从总额度中减去撮合额度
            orderList[orderAddress][ordrePrimaryKey].tradePair['trade'].sellAmount = orderList[orderA
            orderList[orderAddress][ordrePrimaryKey].betweens[betweenPrimaryKey] =
                GoBetween({sellAmount:amount, buyAddress:buyAddress,carryOut:carryOutAddress,randIHas
                        randJHash:randJHash,randKey:randKey, primaryKey:betweenPrimaryKey, buyAmount:
            orderList[orderAddress][ordrePrimaryKey].betweensKeys[ordrePrimaryKey].push(betweenPrimar
            emit betweenEvent(orderAddress,sellToken,buyToken,amount,buyAddress,carryOutAddress,randI
     }

    //  /**
    //  * 执行挂单
    //  * orderAddress     挂单钱包地址
    //  * sellToken        卖出合约地址
    //  * buyToken         卖入合约地址
    //  * 参数参考struct GoBetween
    //  */
     function carryOut( bytes32 randI,  bytes32 randJ, address orderAddress,address payable betweensA
                bytes32 ordrePrimaryKey,bytes32 betweenPrimaryKey) public {
            uint256 blockNo = orderList[orderAddress][ordrePrimaryKey].blockHeight
                .add(orderList[orderAddress][ordrePrimaryKey].betweens[betweenPrimaryKey].blockNo);
            require( block.number < blockNo,"carryOut:blockHeight invalid");

            GoBetween memory between = orderList[orderAddress][ordrePrimaryKey].betweens[betweenPrima

            require(between.carryOut == msg.sender,"carryOut:caller is not the carryOut");
            require(checkHash(randI,between.randIHash),"carryOut:randI invalid");
            require(checkHash(randJ,between.randJHash),"carryOut:randJ invalid");
            require(between.sellAmount > 0,"carryOut:sellAmount not enough");
                //可交易总额度
                uint256 totalAmount = between.sellAmount;
                //计算手续费（手续费= fee / 10000)
                uint256 fee = totalAmount.mul(orderList[orderAddress][ordrePrimaryKey].fee).div(10000
                //计算转给乙方的数量
                uint256 buyAmount = totalAmount.sub(fee);
                //计算撮合者及执行者的收益
                uint256 exchange = fee.div(2);
                //可用额度置0
                orderList[orderAddress][ordrePrimaryKey].betweens[betweenPrimaryKey].sellAmount = 0;
                //给乙方转账

                if(orderList[orderAddress][ordrePrimaryKey].tradePair['trade'].sellToken == address(0
                    between.buyAddress.transfer(buyAmount);
                    msg.sender.transfer(exchange);
                    betweensAddress.transfer(exchange);
                }else{
                    IERC20(orderList[orderAddress][ordrePrimaryKey].tradePair['trade'].sellToken).saf
                    //给撮合者转账
                    IERC20(orderList[orderAddress][ordrePrimaryKey].tradePair['trade'].sellToken).saf
                    //给执行者转账
                    IERC20(orderList[orderAddress][ordrePrimaryKey].tradePair['trade'].sellToken).saf
```

```
                }
                emit withdrawEvent(between.buyAddress,buyAmount);
                emit withdrawEvent(betweensAddress,exchange);
                emit withdrawEvent(msg.sender,exchange);
     }
    // // withdraw
    // /**
    //  * 返回用户代币
    //  */
    function retrace(bytes32 ordrePrimaryKey) public returns(bool) {
            uint256 _amount = orderList[msg.sender][ordrePrimaryKey].tradePair['trade'].sellAmount;
            orderList[msg.sender][ordrePrimaryKey].tradePair['trade'].sellAmount
                = orderList[msg.sender][ordrePrimaryKey].tradePair['trade'].sellAmount.sub(_amount);
                bytes32[] memory keys = orderList[msg.sender][ordrePrimaryKey].betweensKeys[ordrePrim
                for(uint j = 0; j < keys.length; j++){
                    uint256 blockNo = orderList[msg.sender][ordrePrimaryKey].blockHeight
                            .add(orderList[msg.sender][ordrePrimaryKey].betweens[keys[j]].blockNo);
                    if (block.number > blockNo){
                        _amount = _amount.add(orderList[msg.sender][ordrePrimaryKey].betweens[keys[j]
                    }

                }
                if(orderList[msg.sender][ordrePrimaryKey].tradePair['trade'].sellToken == address(0))
                    msg.sender.transfer(_amount);
                }else{
                    IERC20(orderList[msg.sender][ordrePrimaryKey].tradePair['trade'].sellToken).safeT
                }
                emit retraceEvent(msg.sender, _amount);

        return true;
    }
    function checkHash(bytes32 _original, bytes32 hash) public pure returns(bool isEqual){
        bytes32 original = sha256(abi.encodePacked(_original));
        isEqual = (keccak256(abi.encodePacked(original)) == keccak256(abi.encodePacked(hash)));
        return (isEqual);
    }
}
```

Analysis of audit results

Re-Entrancy

- **Description:**
  One of the features of Ethereum smart contracts is the ability to call and utilise code of other external contracts. Contracts also typically handle ether, and as such often send ether to various external user addresses. The operation of calling external contracts, or sending ether to an address, requires the contract to submit an external call. These external calls can be hijacked by attackers whereby they force the contract to execute further code (i.e. through a fallback function) , including calls back into itself. Thus the code execution "re-enters" the contract. Attacks of this kind were used in the infamous DAO hack.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Arithmetic Over/Under Flows

- **Description:**
  The Ethereum Virtual Machine (EVM) specifies fixed-size data types for integers. This means that an integer variable, only has a certain range of numbers it can represent. A uint8 for example, can only store numbers in the range [0,255]. Trying to store 256 into a uint8 will result in 0. If care is not taken, variables in Solidity can be exploited if user input is unchecked and calculations are performed which result in numbers that lie outside the range of the data type that stores them.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unexpected Ether

- **Description:**
  Typically when ether is sent to a contract, it must execute either the fallback function, or another function described in the contract. There are two exceptions to this, where ether can exist in a contract without having executed any code. Contracts which rely on code execution for every ether sent to the contract can be vulnerable to attacks where ether is forcibly sent to a contract.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Delegatecall

- **Description:**
  The CALL and DELEGATECALL opcodes are useful in allowing Ethereum developers to modularise their code. Standard external message calls to contracts are handled by the CALL opcode whereby code is run in the context of the external contract/function. The DELEGATECALL opcode is identical to the standard message call, except that the code executed at the targeted address is run in the context of the calling contract along with the fact that msg.sender and msg.value remain unchanged. This feature enables the implementation of libraries whereby developers can create reusable code for future contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Default Visibilities

- **Description:**
  Functions in Solidity have visibility specifiers which dictate how functions are allowed to be called. The visibility determines whether a function can be called externally by users, by other derived contracts, only internally or only externally. There are four visibility specifiers, which are described in detail in the Solidity Docs. Functions default to public allowing users to call them externally. Incorrect use of visibility specifiers can lead to some devestating vulernabilities in smart contracts as will be discussed in this section.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Entropy Illusion

- **Description:**
  All transactions on the Ethereum blockchain are deterministic state transition operations. Meaning that every transaction modifies the global state of the Ethereum ecosystem and it does so in a calculable way with no uncertainty. This ultimately means that inside the blockchain ecosystem there is no source of entropy or randomness. There is no rand() function in Solidity. Achieving decentralised entropy (randomness) is a well established problem and many ideas have been proposed to address this (see for example, RandDAO or using a chain of Hashes as described by Vitalik in this post).
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

External Contract Referencing

- **Description:**
  One of the benefits of the Ethereum global computer is the ability to re-use code and interact with contracts already deployed on the network. As a result, a large number of contracts reference external contracts and in general operation use external message calls to interact with these contracts. These external message calls can mask malicious actors intentions in some non-obvious ways, which we will discuss.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unsolved TODO comments

- **Description:**
  Check for Unsolved TODO comments
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Short Address/Parameter Attack

- **Description:**
  This attack is not specifically performed on Solidity contracts themselves but on third party applications that may interact with them. I add this attack for completeness and to be aware of how parameters can be manipulated in contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unchecked CALL Return Values

- **Description:**
  There a number of ways of performing external calls in solidity. Sending ether to external accounts is commonly performed via the transfer() method. However, the send() function can also be used and, for more versatile external calls, the CALL opcode can be directly employed in solidity. The call() and send() functions return a boolean indicating if the call succeeded or failed. Thus these functions have a simple caveat, in that the transaction that executes these functions will not revert if the external call (intialised by call() or send()) fails, rather the call() or send() will simply return false. A common pitfall arises when the return value is not checked, rather the developer expects a revert to occur.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Race Conditions / Front Running

- **Description:**
  The combination of external calls to other contracts and the multi-user nature of the underlying blockchain gives rise to a variety of potential Solidity pitfalls whereby users race code execution to obtain unexpected states. Re-Entrancy is one example of such a race condition. In this section we will talk more generally about different kinds of race conditions that can occur on the Ethereum blockchain. There is a variety of good posts on this subject, a few are: Ethereum Wiki - Safety, DASP - Front-Running and the Consensus - Smart Contract Best Practices.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Denial Of Service (DOS)

- **Description:**
  This category is very broad, but fundamentally consists of attacks where users can leave the contract inoperable for a small period of time, or in some cases, permanently. This can trap ether in these contracts forever, as was the case with the Second Parity MultiSig hack
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Block Timestamp Manipulation

- **Description:**
  Block timestamps have historically been used for a variety of applications, such as entropy for random numbers (see the Entropy Illusion section for further details), locking funds for periods of time and various state-changing conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Constructors with Care

- **Description:**
  Constructors are special functions which often perform critical, privileged tasks when initialising contracts. Before solidity v0.4.22 constructors were defined as functions that had the same name as the contract that contained them. Thus, when a contract name gets changed in development, if the constructor name isn't changed, it becomes a normal, callable function. As you can imagine, this can (and has) lead to some interesting contract hacks.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unintialised Storage Pointers

- **Description:**
  The EVM stores data either as storage or as memory. Understanding exactly how this is done and the default types for local variables of functions is highly recommended when developing contracts. This is because it is possible to produce vulnerable contracts by inappropriately intialising variables.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Floating Points and Numerical Precision

- **Description:**
  As of this writing (Solidity v0.4.24), fixed point or floating point numbers are not supported. This means that floating point representations must be made with the integer types in Solidity. This can lead to errors/vulnerabilities if not implemented correctly.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

tx.origin Authentication

- **Description:**
  Solidity has a global variable, tx.origin which traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in smart contracts leaves the contract vulnerable to a phishing-like attack.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

armors.io

contact@armors.io